COOLEY GODWARD LLP
ATTORNEY DOCKET NO.: CNTW-008/00US
CLIENT NO.: 036958-2011

EXPRESS MAIL NO. EL 821812973 US

APPLICATION FOR PATENT

TITLE: SYSTEM AND METHOD FOR MODELING A NETWORK DEVICE'S CONFIGURATION

INVENTOR(S): MIKE COURTNEY

RELATED APPLICATIONS

[0001] The present application is related to commonly owned and assigned application nos.:

> 09/730,864 entitled *System and Method for Configuration, Management and Monitoring of Network Resources*, filed December 6, 2000;
>
> 09/730,680 entitled *System and Method for Redirecting Data Generated by Network Devices*, filed December 6, 2000;
>
> 09/730,863 entitled *Event Manager for Network Operating System*, filed December 6, 2000;
>
> 09/730,671 entitled *Dynamic Configuration of Network Devices to Enable Data Transfers*, filed December 6, 2000;
>
> 09/730,682 entitled *Network Operating System Data Directory*, filed December 6, 2000; and
>
> 09/799,579 entitled *Global GUI Interface for Network OS*, filed March 6, 2001;

all of which are incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention relates to network device configuration. In particular, but not by way of limitation, the present invention relates to systems and methods for retrieving configurations from network devices and generating corresponding command models.

BACKGROUND OF THE INVENTION

[0003] Networks, and in particular, the Internet, have revolutionized communications. Data vital to the continued prosperity of the world economy is constantly being exchanged between end-users over these networks. Unfortunately, the expansion and maintenance of present networks is outpaced by the demand for additional bandwidth. Network equipment is often difficult to configure, and qualified network engineers are in extremely short supply. Thus, many needed network expansions and upgrades must be delayed until these engineers are available. While these upgrades and expansions are pending, end-users continue to suffer poor network performance.

[0004] Cisco™ routers, for example, are notoriously difficult to configure--especially in light of the new XML-based interfaces introduced by competitors such as Juniper Networks™. Instead of a user-friendly XML-based interface, Cisco™ uses a cumbersome command line interface (CLI) for its routers. Cisco's™ CLI is the result of many years of semi-controlled modifications to its router operating systems and has resulted in a tangled mess of commands and subcommands. This cumbersome interface is one reason that Cisco™ requires that Cisco-certified engineers work on its routers.

[0005] Cisco™ could reduce the complexity of its routers and reduce the need for Cisco-certified engineers by producing a user-friendly interface. If Cisco™ attempted to abandon its CLI in favor of such a user-friendly interface, however, many years of development and expertise could be lost. Moreover, even if it could develop a user-friendly interface, there is presently no economical way to integrate it into the thousands

of existing Cisco™ routers. Despite the difficulties in implementing a more user-friendly interface, to remain competitive, Cisco™ and similarly situated companies need to move away from their present interfaces. Present technology, however, does not provide these companies with an acceptable option that allows continued use of their extensive interface knowledge base while simultaneously providing system administrators and network engineers with a user-friendly interface. Moreover, present technologies do not provide an acceptable way to provide backward compatibility of new user-friendly interfaces with existing network devices.

[0006] Cisco™, of course, is not the only network device manufacturer to face this interface-upgrade problem. Many manufacturers would like to continue using their existing interface knowledge base while providing system administrators a user-friendly, consistent interface. Accordingly, a system and method are needed that will allow manufacturers, like Cisco™, to create user-friendly interfaces for both next-generation and existing devices.

SUMMARY OF THE INVENTION

[0007] Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous

modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

[0008] In one embodiment, for example, the present invention can provide a system and method for modeling the configuration of a network device. Such a system could include a CLI-to-XML converter connected to a schema storage device or a CLI-to-XML converter in combination with a document object model (DOM) generator. Other embodiments could include, for example, a CLI-to-XML converter, a schema hash system, and a DOM generator.

[0009] In operation, one embodiment of the present invention can model a network device's configuration by retrieving a the network device's configuration, in a native format, from the network device--or an alternate location--and converting it into a standard-format configuration such as an XML document or a DOM. This standard-format configuration provides system administrators with an easy-to-use, familiar device configuration format for different network devices. That is, instead of being forced to manipulate a difficult CLI-based configuration format, or other format system administrators can use the standard-format configuration to interact with the target network device. Moreover, one embodiment of the present invention can allow system administrators to use the same standard configuration format across multiple brands and models of network devices. Thus, in networks that employ multiple brands and models of network devices, system administrators can be presented with similar configuration

formats for each device despite the fact that the native configuration formats for the different devices are significantly different.

[0010] The process for actually converting a native-format configuration for a network device into a standard-format configuration is generally a multi-step process. For example, one embodiment of the present invention initially determines the target network device's characteristics such as manufacturer, model, operating system version, etc. Next, using some or all of this characteristic information, an appropriate configuration schema can be retrieved from a schema storage device. Briefly, the schema can include a standard representation of the command structure for a particular type of network device. For example, one schema could contain a representation of the command structure for all model 7500 Cisco™ routers using OS version 12.1, and another schema could contain a representation of the command structure routers using OS version 12.2. The schema, its creation, and its use are fully described in commonly owned and assigned U.S. patent application no. ______________, Attorney Docket No. CNTW-007/US, entitled *System and Method for Generating a Configuration Schema*, which is incorporated herein by reference.

[0011] In certain embodiments, this schema can be directly used to generate an XML document that represents the configuration of the particular network device. In the presently preferred embodiment, however, an intermediate representation, e.g., a hash representation, of the schema is generated and the intermediate representation is used to more quickly generate the corresponding XML document. By using the intermediate

representation, the number of instruction cycles needed to generate the XML document is reduced significantly when compared to generating the XML document directly.

[0012] To actually assemble an XML document, one embodiment of the present invention generates an XML representation of each native-format command in the network device's configuration by associating each command with the schema, or its hash representation. The XML document itself can be used to represent the standard-format configuration, or alternatively, the XML document can be converted into a DOM, and the DOM can represent the standard-format configuration. Notably, the integrity of the generated DOM can be verified via the schema that was used to generate the XML document, thereby providing a "closed-loop" capability.

[0013] As previously stated, the above-described embodiments and implementations are for illustration purposes only. Numerous other embodiments, implementations, and details of the invention are easily recognized by those of skill in the art from the following descriptions and claims.

<u>BRIEF DESCRIPTION OF THE DRAWINGS</u>

[0014] Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings wherein:

FIGURE 1 is a block diagram of a conventional network;

FIGURE 2 is a block diagram of a conventional router;

FIGURE 3 is a block diagram of one embodiment of a system constructed in accordance with the principles of the present invention;

FIGURE 4 is a block diagram of an alternate embodiment of a system constructed in accordance with the principles of the present invention;

FIGURE 5 is a block diagram of one implementation of the DOM generator shown in FIGURE 3;

FIGURE 6 is a flowchart of one method for operating the DOM generator shown in FIGURE 5; and

FIGURE 7 is a flowchart of one method for generating an intermediate representation described with relation to FIGURE 6.

DETAILED DESCRIPTION

[0015] Referring now to the drawings, where like or similar elements are designated with identical reference numerals throughout the several views, and referring in particular to FIGURE 1, it illustrates a block diagram of a conventional network system 100. In this network system 100, end-users 105 are connected to servers 110, which are connected to networking equipment such as hubs, not shown, optical components 115, and routers 120. Using the networking equipment, end-users 105 that are associated with different servers 110 can exchange data.

[0016] As new servers 110 and end-users 105 are added to the overall system 100, or as new software becomes available, the routers 120 and/or optical components 115 of the

network system 100 may need reconfiguring. To reconfigure these components, a system administrator 125--with the proper authorization--could access the router 120 and/or optical component 115 by, for example, establishing a telnet connection to the component and transferring configuration instructions thereto.

[0017] Referring now to FIGURE 2, it is a block diagram of one type of conventional router. In this representation, a processor 125 is connected to a configuration interface 130, an operating system (OS) storage module 135, a command storage module 140, a configuration storage module 145, and a routing module 150. The illustrated arrangement of these components is logical and not meant to be an actual hardware diagram. Thus, the components can be combined or further separated in an actual implementation. Moreover, the construction of each individual component is well-known to those of skill in the art.

[0018] Still referring to FIGURE 2, when a system administrator 125 wishes to reconfigure a router 120, he accesses the router 120 through the configuration interface 130 and retrieves the present configuration for the router 120 from the configuration storage module 145. If necessary, the system administrator 125 can review available configuration commands and associated bounds by accessing and reviewing the commands stored in the command storage module 140. In essence, the command storage module 140 provides the knowledge base for a "help" screen. The commands stored in the command storage module 140 are often unique to the particular OS version stored in the OS module 135.

[0019] After the system administrator 125 has assembled the new configuration instructions, these instructions are pushed through the configuration interface 130 and stored in the configuration storage module 145. As previously described, for Cisco™ routers, interaction is generally through a CLI. In other words, the command storage module 140 is queried through the CLI; available commands are returned through the CLI; and new configuration commands are provided to the router 120 through the CLI. Unfortunately, the CLI is difficult to manage and requires highly skilled engineers for even simple tasks.

[0020] For other routers, the configuration interface 130 could be XML based. Although the XML-based interface is easier to navigate than a CLI, each network device manufacturer that uses an XML-based interface generally structures its interface in a proprietary fashion. Thus, network engineers are still forced to learn many different interfaces and command structures even for XML-based network devices.

[0021] Referring now to FIGURE 3, it is a block diagram of one embodiment of a system constructed in accordance with the principles of the present invention. In this embodiment, a DOM generator 160, which is more fully described with relation to FIGURE 5, is connected to a network device 165, a schema storage device 170, a system administrator 175, a DOM storage device 180, and various DOM applications 185, which will be discussed in more detail below.

[0022] In one method of operation, the system administrator 175 initially notifies the DOM generator 160 to model the configuration for the network device 165. In other words, the DOM generator 160 is instructed to convert the active command format for the network device 165 into an XML and/or DOM format. In response, the DOM generator 160 either polls the network device 165 to discover the device's characteristics, e.g., manufacturer, model, operating system version, etc., or retrieves the information from a database (not shown). Next, the DOM generator 160 identifies and retrieves, from the schema storage device 170, the schema corresponding to the device characteristics for the network device 165. The DOM generator 160 then retrieves the configuration from the network device 165 and, using the retrieved schema, converts the individual commands of the configuration into a DOM. The resulting DOM can then be stored in the DOM storage device 180 in association with an identifier for the network device 165. Note that storage devices 170 and 180 could, in fact, be integrated into a single device.

[0023] One advantage of the DOM format is that it provides a standard format for most network device configurations. Generally, applications that use or manipulate network device configurations must be customized for each manufacturer, each model, and each OS version. This type of customization often requires many different versions of the same application. By converting each network device's configuration into a DOM format, however, applications can be designed to utilize a single, standard configuration format and thereby limit the need for customizations.

[0024] Although many different types of applications can utilize a DOM, a select few are represented in FIGURE 3 as DOM applications. For example, one such application is a DOM-based graphical user interface (GUI) 190. In this application, the hashed schema and/or the resulting DOM instance are used to drive the GUI used by the system administrator 175. The advantage of such a GUI 190 is that the system administrator 175 is presented with network device configurations in a standard, consistent format regardless of the characteristics of the particular network device.

[0025] Another application that utilizes the DOM is the XML-XML converter 195, also called the standard XML-to-native XML converter. As previously described, some network devices include XML-based interfaces. However, these XML-based interfaces are generally based on proprietary (native) configuration instructions. Thus, the system administrator 175 may interface with one XML-based network device in a very different way than another XML-based network device. To standardize the interface between these various XML-based network devices, the XML-XML converter converts a standard XML-based instruction into a native XML-based instruction. In other words, the XML-XML converter allows the system administrator 175 to use the same XML-based command format for most network devices even though each device may require its own native XML-based command format.

[0026] Like the XML-XML converter 195, the XML-CLI converter 200 allows the system administrator 175 to interface with CLI-based network devices using a standard XML-based command format instead of a CLI-based command format. Other DOM-

based applications may include lightweight directory access protocol (LDAP) for storing and manipulating schema, hash representations, and device configuration commands. These converters convert XML-based configurations into a LDAP-based configuration and LDAP-based configurations into XML-based configurations.

[0027] Yet another possible DOM application is the comparator 210, which is configurable to identify the differences between two DOMs. For example, if the configuration for a target network device were changed, the new configuration could be retrieved from the device and converted to a DOM. The comparator 210 could then compare the new DOM against the original DOM to thereby identify any changes, additions, and/or deletions. The comparator can then record these changes in a markup DOM using a configuration change markup language and make the markup DOM available to the system administrator for configuration and validation purposes.

[0028] In another embodiment of the comparator 210, the old DOM is compared against a draft DOM instead of a new DOM. In other words, the system administrator 175 generates a draft configuration for a target network device 165. This draft configuration is converted into a DOM, and the comparator 210 compares it against the target network device's original DOM. The system administrator 175 can use this embodiment of the comparator to view the configuration changes before the draft DOM is finalized and pushed to the target network device 165.

[0029] The DOM applications can also include an (API) application programming interface 215. This API provides a mechanism whereby the DOM can be transferred to/from other software programs, which may reside on network devices. Accordingly, the DOM can be programmatically modified outside of the embodiment and resubmitted.

[0030] Referring now to FIGURE 4, it is a block diagram of an alternate embodiment of a system 220 constructed in accordance with the principles of the present invention. In this embodiment, the DOM generator 160 is connected through a network 225 to the network devices 165, the system administrator 175, the schema storage device 170, and the DOM applications 180. This embodiment illustrates that the components described herein can be distributed in a number of ways and without impacting the basic operation of this system as described with regard to FIGURE 3.

[0031] Referring now to FIGURE 5, it is a block diagram of one implementation of the DOM generator 160 shown in FIGURE 3. In this embodiment, the DOM generator 160 includes a schema hash system 230, an XML converter 235, and a DOM transformer 250. These components can be connected to the schema storage device 170, the target network device 165, a DOM storage device 245 and an XML storage device 250.

[0032] In this embodiment, the XML converter 235, using the appropriate schema, generates an XML document containing an XML representation of the network device's configuration. This XML document is then passed to the DOM transformer 240, which converts the XML document into a DOM. The output from the XML converter 235

and/or the DOM transformer 240 can be stored and passed to relevant software applications. For example, the output from the XML converter 235 can be stored in the XML storage device 250 and the output from the DOM transformer 240 can be stored in the DOM storage device 245.

[0033] Notably, the XML converter 235 of this embodiment can convert the native configuration of the network device 165 into an XML document using an intermediate representation of the schema associated with the network device 165, such as a hash table generated by the hash system 230, instead of the schema itself. By using an intermediate representation of the appropriate schema, the XML converter 235 can reduce the time and processing requirements needed to convert a native configuration into a corresponding XML document. The creation and use of the intermediate representation is described more fully with regard to FIGURE 6.

[0034] The operation of the DOM generator 160 can be further illustrated by reference to the flowchart in FIGURE 6. As depicted, the DOM generator 160 determines the target network device's characteristics by polling the network device or accessing a database (not shown) containing such information (step 255). Next, the XML converter 235 identifies the appropriate intermediate representation for the target network device 165 (step 260). As previously described, this intermediate representation provides the necessary data to convert the native-format configuration of the target network device 165 into a standard format such as an XML format.

[0035] Possibly concurrently with the XML converter 235 identifying the corresponding intermediate representation, the XML converter 235 retrieves the configuration from the network device 165 and identifies each initial command within each configuration line (steps 265 and 270). For example, the XML converter 235 could locate command distinguishing tags embedded in the configuration such as "begin command" and/or "end command." Alternatively, the XML converter 235 could use logical indicators within the configuration to distinguish the individual commands. Either way, using the identified initial command, the XML converter 235 generates a look-up key that is used to index the hash table, locate a hash map object that corresponds to the look-up key and retrieve that hash map object (steps 275 and 280). The hash map object contains schema information regarding the command or value such as whether optional or required data type, etc. Finally, using this hash map object, the XML converter 235 can assemble the XML-based command and write it to the corresponding XML document (step 295).

[0036] The above process should be repeated for each command in the network device's native-format configuration. With regard to FIGURE 6, this process is represented by determining whether any more commands need to be converted (step 300). If so, branch 305 is followed to step 270 and a next native-format command is identified. The process for this command is then repeated. If, on the other hand, all native-format commands have been converted, branch 310 is followed and the XML converter 235 assembles all of the generated XML commands into an XML document that can be stored in the XML storage device and/or provided to the DOM transformer 240 (step 315).

[0037] Once the XML document has been assembled, it can be passed to the DOM transformer 240 where a DOM corresponding to the XML document can be generated (step 320). The process for converting an XML document to a DOM is well known in the art and, thus, not described here. Notably, the DOM transformer 240 can verify its transformation process against the appropriate schema stored in the schema storage device 170 (step 325). In other words, each configuration command in the DOM should have a particular format, which are defined by the configuration schema corresponding to the target network device 165. Thus, the DOM transformer 240 can compare the generated DOM against the corresponding configuration schema to verify that the DOM was properly constructed.

[0038] Referring now to FIGURE 7, it is a flowchart of one method for generating an intermediate representation of a configuration schema. In this embodiment, a command is initially retrieved from the previously assembled configuration schema (step 328). Additionally, any related higher-level commands (called parent commands) in the configuration schema can be retrieved (step 330). The retrieved command and the retrieved parent commands can then be used to generate a unique hash key for the retrieved command (step 330).

[0039] After the unique hash key is generated, a corresponding hash object can also be generated. This hash object can include basic information related to the generated hash key. To generate the hash object, information such as data type, sibling commands, and application specific information is retrieved and assembled into the schema object (steps

335 and 340). The data type information, for example, can indicate whether the data associated with a particular command is a string, an integer, etc. and the sibling information can identify commands at the same hierarchical level as the initially retrieved command that have the same parent command as the initially retrieved command. Additionally, in certain embodiments, specialized application information can also be retrieved (step 345). This application information, for example, can define special processing requirements for a schema.

[0040] Once the relevant information has been collected, the corresponding schema object can be assembled and the hash map assembled for the unique key and schema object (step 350 and 355). If there are any more commands in the schema that need to be modeled, branch 362 is followed and the next command can be retrieved (step 328). If all of the commands have been modeled, then branch 364 can be followed and the various hash objects can be stored as a completed hash table (step 365).